Stochastic Gradient Descent

STAT 672 Project

Tom Wallace

George Mason University

Spring 2018

Optimization is fundamental to statistical modeling

Suppose that we have a typical supervised classification problem

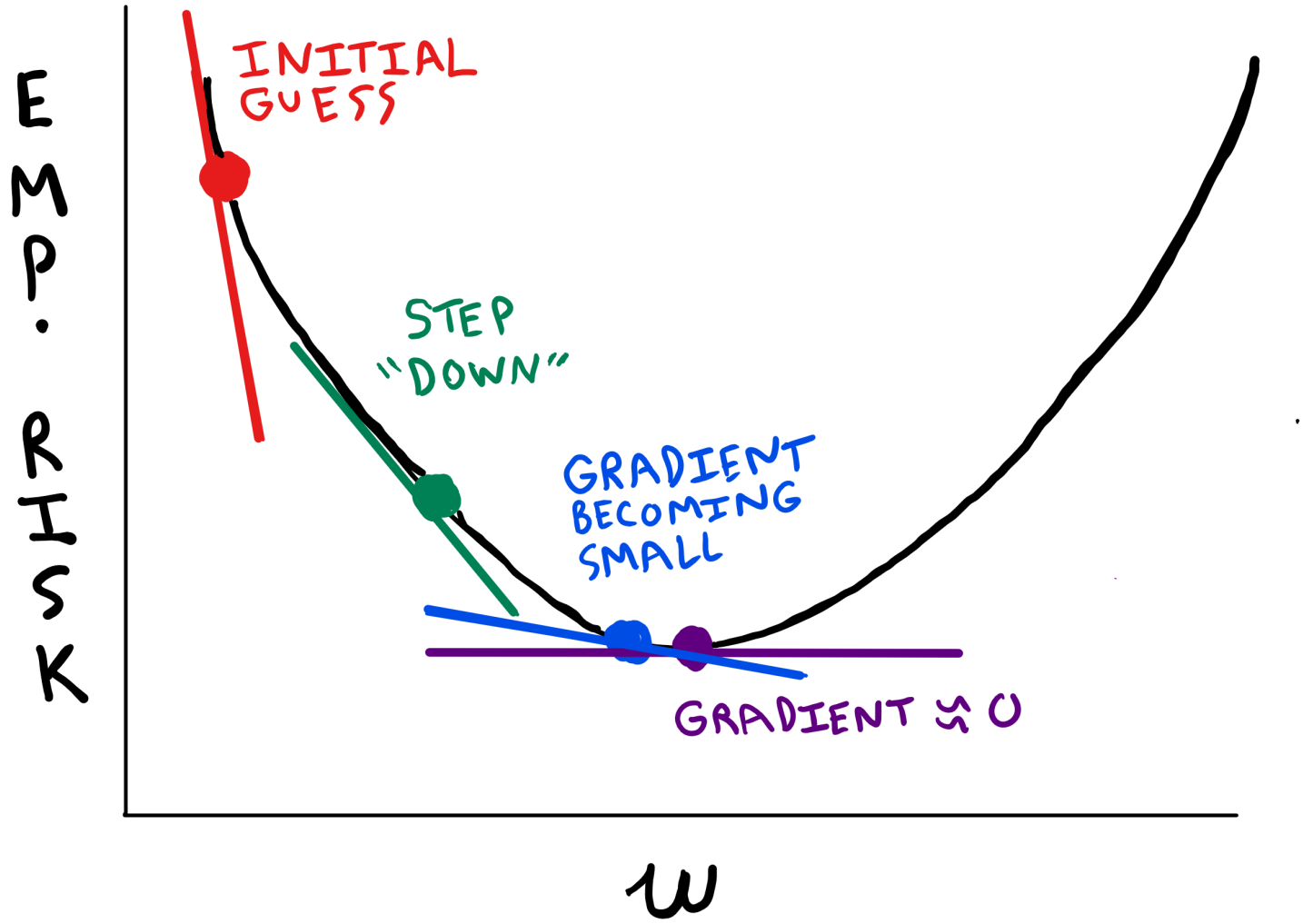
- Non-parametric: no assumptions about distribution of data
- Feature vector $\mathbf{X}_i$, label Y_i
- Want to find best prediction function f_w^* from class $\mathcal{F}$
- Optimization: pick weights $\hat{\mathbf{w}}$ that minimize empirical risk according to some convex loss function $L(f_w(x), y)$

Our lack of assumptions takes away some familiar tools for finding $\hat{\mathbf{w}}$

- Cannot analytically identify $\hat{\mathbf{w}}$ (e.g. in OLS $= (X'X)^{-1}X'Y)$)
- Newton-Raphson requires 2nd derivative of L, which might be a pain (unlike in, for example, GLM)

But we still know that if L is convex, there is a unique global minimum, and the gradient at that point must be 0

Gradient descent is an iterative search procedure



A more formal explanation

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w L(f_w(\mathbf{X}_i), y_i)$$

Stop if gradient $\leq \epsilon$

Step size γ can vary over time

Theoretical guarantees on speed of convergence ($\rho :=$ size of error):

- Version presented here: $-\log \rho \sim t$
- More optimized version: $-\log \log \rho \sim t$

Batch gradient descent is computationally expensive

In "plain" (batch) gradient descent, for every step, we have to evaluate the gradient at every observation

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w L(f_w(\mathbf{X}_i), y_i)$$

This becomes computationally intractable as n grows large

- If we have $n = 10$ million, we have to evaluate the gradient 10 million times for every step (and the algorithm may take many steps to converge)
- May take hours or days to converge

Stochastic gradient descent (SGD) takes less time

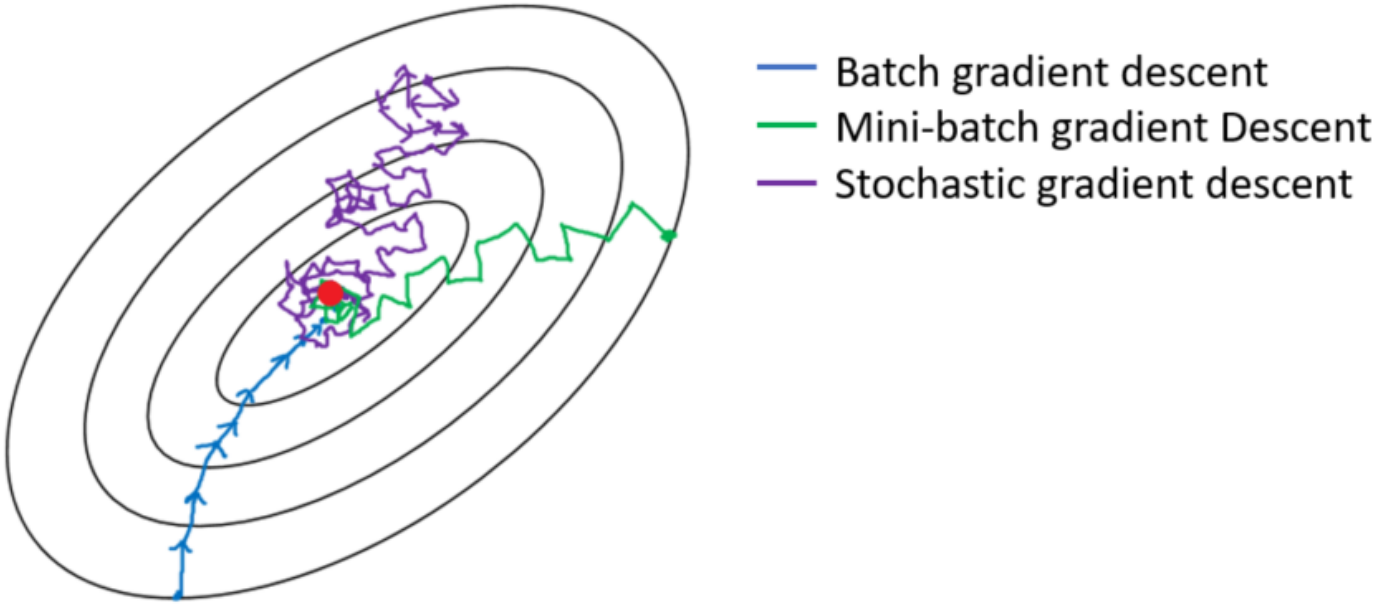
For each step, gradient is computed for a single randomly chosen observation i:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \nabla_w L(f_w(\mathbf{X}_i), y_{i)}$$

This simplification makes approximation much "noisier", and hence SGD requires more iterations

But, each iteration is faster and so SGD can reach a predefined level of risk or error in less time

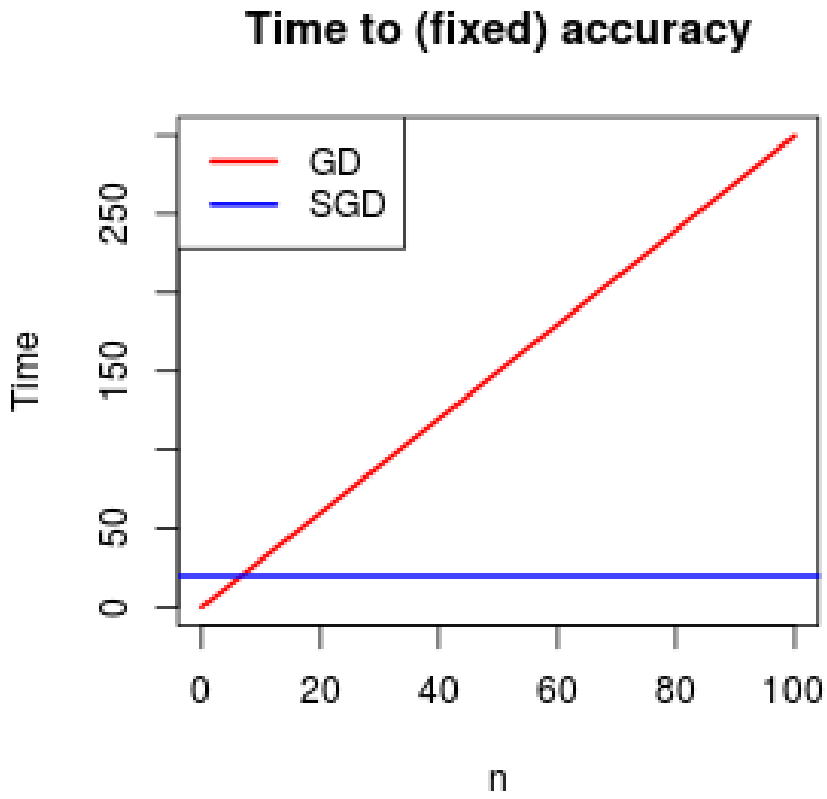
SGD is noisier than batch GD



Source: towardsdatascience.com

SGD is useful when n is large & compute time is important

	GD	**SGD**
Time to accuracy ρ	$n \log \frac{1}{\rho}$	$\frac{1}{\rho}$



SGD is widely used in industry

If a Silicon Valley press release uses any of the following phrases...

- "Neural networks"
- "Machine learning"
- "AI"

...SGD probably is involved.

Example: Google's **AlphaGo** program.


nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that
can beat a champion Go player PAGE 484
ALL SYSTEMS GO

Google slides from ICML 2016

Supervised learning of policy networks

Policy network: 12 layer convolutional neural network

Training data: 30M positions from human expert games (KGS 5+ dan)

Training algorithm: maximise likelihood by stochastic gradient descent

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

Training time: 4 weeks on 50 GPUs using Google Cloud

Results: 57% accuracy on held out test data (state-of-the art was 44%)

Google DeepMind